

TACC Stats I/O Performance Monitoring for the Intransigent

John Hammond, TACC
jhammond@tacc.utexas.edu

IASDS11

Collaborators

- Bill Barth, TACC
 - Consumer, first user
- John McCalpin, TACC
 - Performance guru

Motivation

Thousands of different users

Hundreds of codes

Users don't (in general) profile their codes

NUMA understanding is low

Parallel I/O is hard

Goals

- Monitor all jobs
 - Lightweight
 - No user intervention required
- Send up red flags
- Rank jobs by severity of the flag
- Increase the overall code performance

Approach

- Systematically collect data
- Tag with current job id
- Data mine for better site understanding
- Send report cards to users
- Contact users with problematic jobs

Approach

- Tried to modify sar (sysstat)
 - Fixed interval, not invokable
 - Serializes to custom format
 - Not job oriented
 - Unaware of Lustre, IB, newer Linux counters

- `tacc_stats` is a single exe.
- Runs:
 - at job startup (batch sched. prolog)
 - every 10 minutes (cron)
 - at job shutdown (batch sched. epilog)
- Collects data on each node
- Data stored locally
- Collected nightly to a central location

Data Collected

- current time, job id, host uptime
- core and socket perf. counters
- IB perf. counters
- Lustre (llite, mdc, osc) statistics
- OS memory, process statistics
- network, block device counters

Performance counters

- AMD64 – Opteron, per core
 - FLOPS, DCSF, UC, DRAM, HT
- Intel – Nehalem, per core
- Intel uncore – Nehalem, per socket
- Program /dev/cpu/*/msr in prolog
- Don't interfere with user

InfiniBand

`ib` – First attempt, HCA counters

32-bit saturating. WTF?

`ib_sw` – Query extended counters on
switch port connected to HCA, swap
rx, tx. Ranger.

`ib_ext` – 64-bit. Lonestar.

All use libibmad.

Lustre Ilite (VFS)

File operation counters

open, close, read, write, seek, flock, ...

read(), write() tallied by bytes requested not returned
L-333, landed

Directory operations missing

create, lookup, (un)link, rename, {mk,rm,read}dir, ...
LU-673

Lustre mdc

RPC opcode oriented

close, getattr, setattr, ...

Missing operations

MDS_REINT

REINT_{CREATE,UNLINK,...}

LDLM_ENQUEUE

LDLM_{IBITS,FLOCK,...}

Lustre osc

RPC opcode based + I/O counters

Overcounting on error.

Must read 300 (#OST) files to get
bytes on the wire for /scratch

1800 system calls (stdio), 0.037 s.

LU-334: Add OSC I/O counters to llite.

Linux memory

Per NUMA node (socket)

Total, Free, Used, Cached, Mapped,
Anon, Page Tables, Slab, ...

Not totally clear what's what.

Linux process/scheduler

Processes, context switches, forks

1, 5, 15 minute load averages

Ticks (per core) in user, nice, system,
sleep, iowait, irq, softirq, idle.

Linux counters, continued

Block device (boring).

Network interfaces.

SysV shared memory.

General VFS.

VM/paging stats.

Weird NUMA allocation stuff.

TACC Stats design

KISS.

Strip-down.

Optimize for small /sys, /proc files.

Collect raw counter values.

Intermediate stats files.

Flat text, awk-able.

Self-describing, schema with names, types, units.

Highly modular, disable types at build or run time.

230 main.c
283 dict.c
184 collect.c
119 schema.c
166 stats.c
253 stats_file.c

130 obd_to_mnt.c
147 llite.c
129 mdc.c
132 osc.c
53 lnet.c

177 ib_ext.c

75 block.c
93 cpu.c
134 mem.c
120 net.c
69 numa.c
113 ps.c
84 sysv_shm.c
62 tmpfs.c
78 vfs.c
46 vm.c

238 amd64_pmc.c

322 intel_pmc3.c
331 intel_uncore.c

```

#define KEYS \
X(rd_ios, "E", "read requests processed"), \
X(rd_merges, "E", "read requests merged with in-queue \
requests"), \
X(rd_sectors, "E,U=512B", "sectors read"), \
X(rd_ticks, "E,U=ms", "wait time for read requests"), \
X(wr_ios, "E", "write requests processed"), \
X(wr_merges, "E", "write requests merged with in-queue \
requests"), \
X(wr_sectors, "E,U=512B", "sectors written"), \
X(wr_ticks, "E,U=ms", "wait time for write requests"), \
X(in_flight, "", "requests in flight"), \
X(io_ticks, "E,U=ms", "time active"), \
X(time_in_queue, "E,U=ms", "wait time for all requests")

static void collect_block_dev(struct stats_type *type, const char *dev)
{
    struct stats *stats = get_current_stats(type, dev);
    if (stats == NULL)
        return;

    char path[80];
    snprintf(path, sizeof(path), "/sys/block/%s/stat", dev);

#define X(k,r...) #k
    collect_key_list(stats, path, KEYS, NULL);
#undef X
}

```

```
static void collect_block(struct stats_type *type)
{
    const char *path = "/sys/block";
    DIR *dir = opendir(path);
    if (dir == NULL) {
        ERROR("cannot open `%s': %m\n", path);
        return;
    }

    struct dirent *ent;
    while ((ent = readdir(dir)) != NULL) {
        if (ent->d_name[0] == '.')
            continue;
        collect_block_dev(type, ent->d_name);
    }

    if (dir != NULL)
        closedir(dir);
}

struct stats_type block_stats_type = {
    .st_name = "block",
    .st_collect = &collect_block,
#define X SCHEMA_DEF
    .st_schema_def = JOIN(KEYS),
#undef X
};
```

```
$tacc_stats 1.0.2
$hostname i101-101.ranger.tacc.utexas.edu
$uname Linux x86_64 2.6.18-194.32.1.el5_TACC \
#18 SMP Mon Mar 14 22:24:19 CDT 2011
$uptime 2689049
!block rd_ios,E rd_merges,E rd_sectors,E,U=512B \
rd_ticks,E,U=ms wr_ios,E wr_merges,E \
wr_sectors,E,U=512B wr_ticks,E,U=ms in_flight \
io_ticks,E,U=ms time_in_queue,E,U=ms
...
```

```
1316840401 2129327
block hda 30477 2055 1826743 36044 53333 3680 \
803120 1083072 0 425308 1119012
...
```

Data Sizes

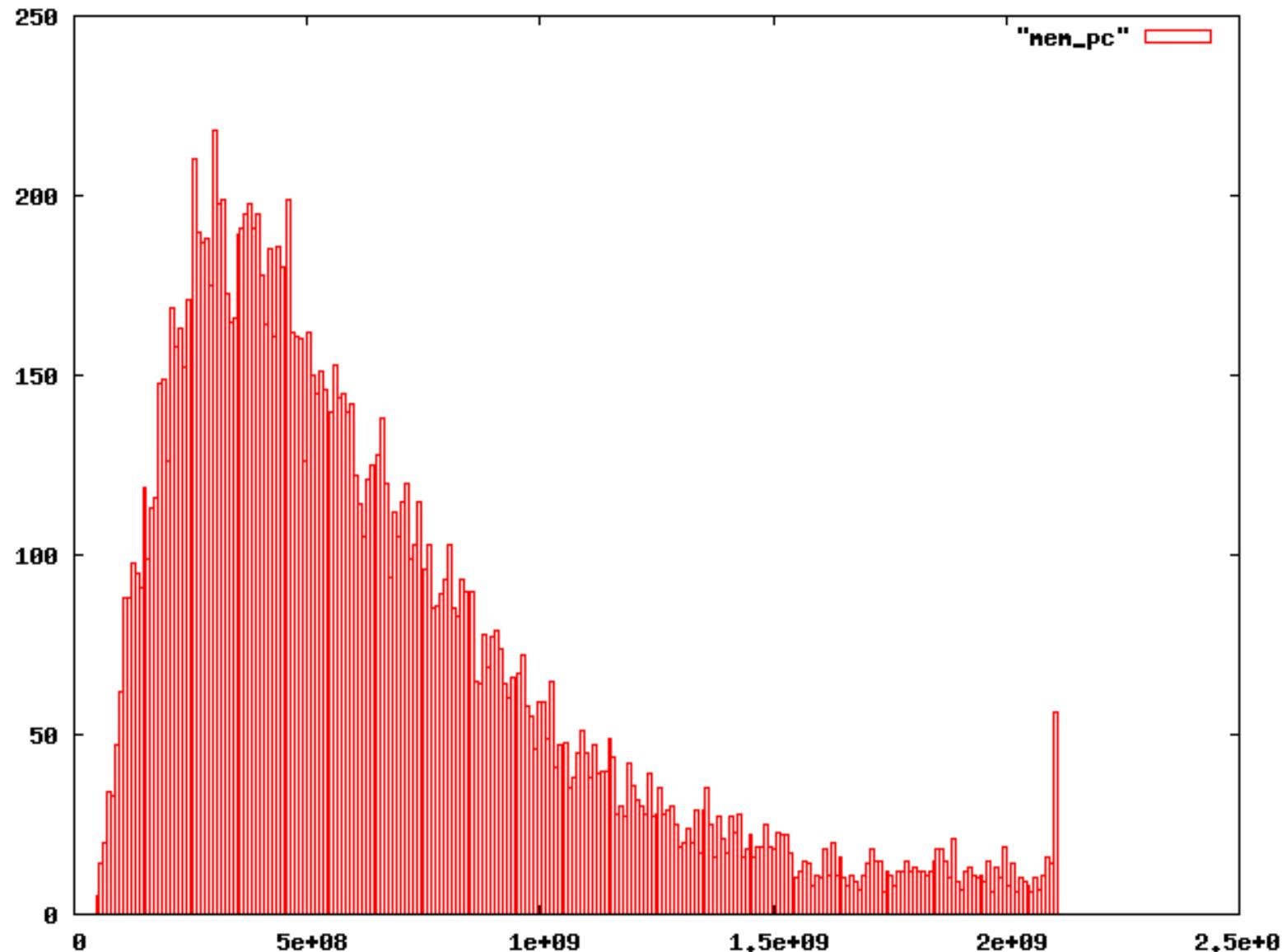
1 record: 3.7 KB, ~545 values

1 file (1 node day): ~160 records = 0.5+ MB
10K lines
180 KB compressed

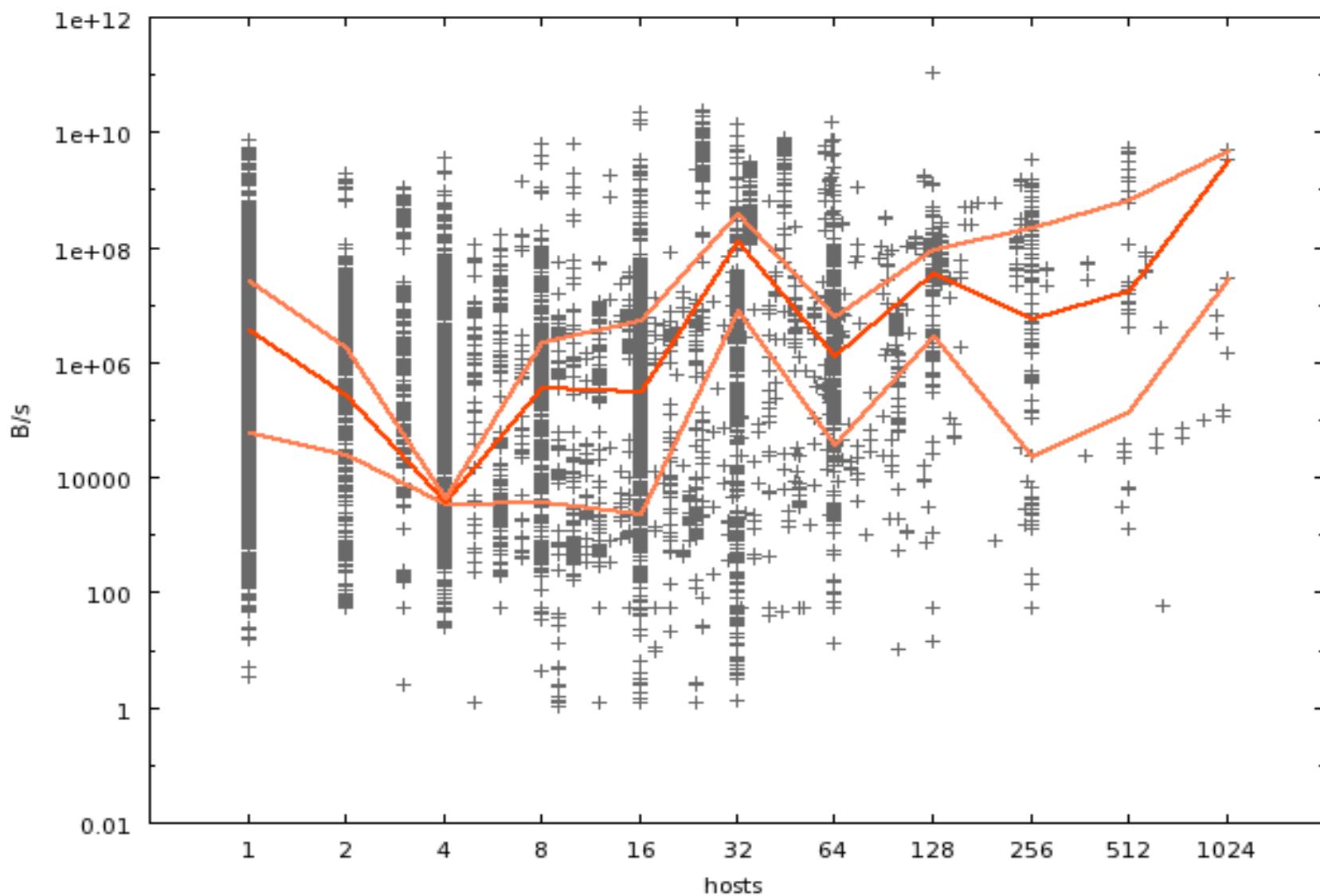
1 node month: 16+ MB

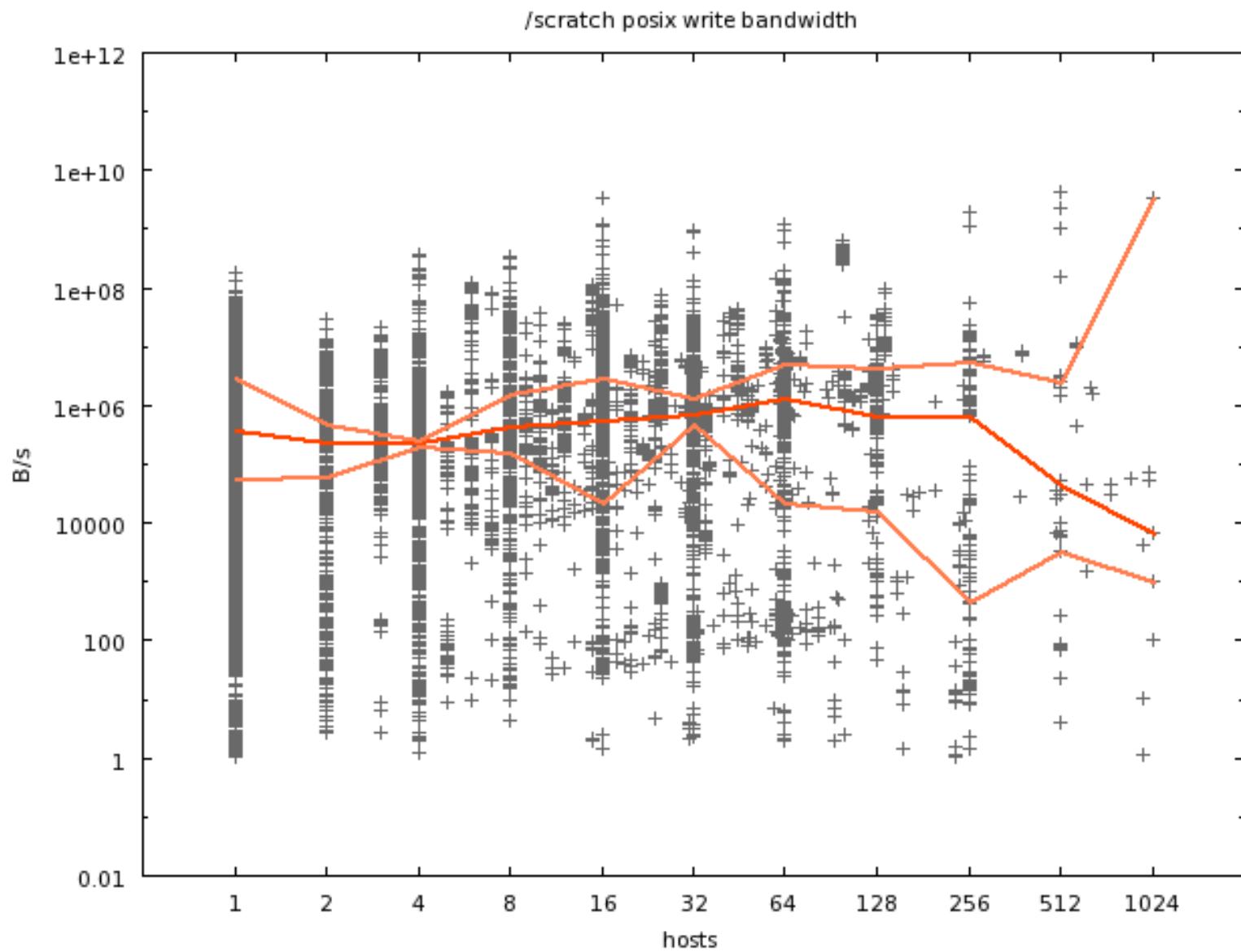
1 Ranger day: 2 GB

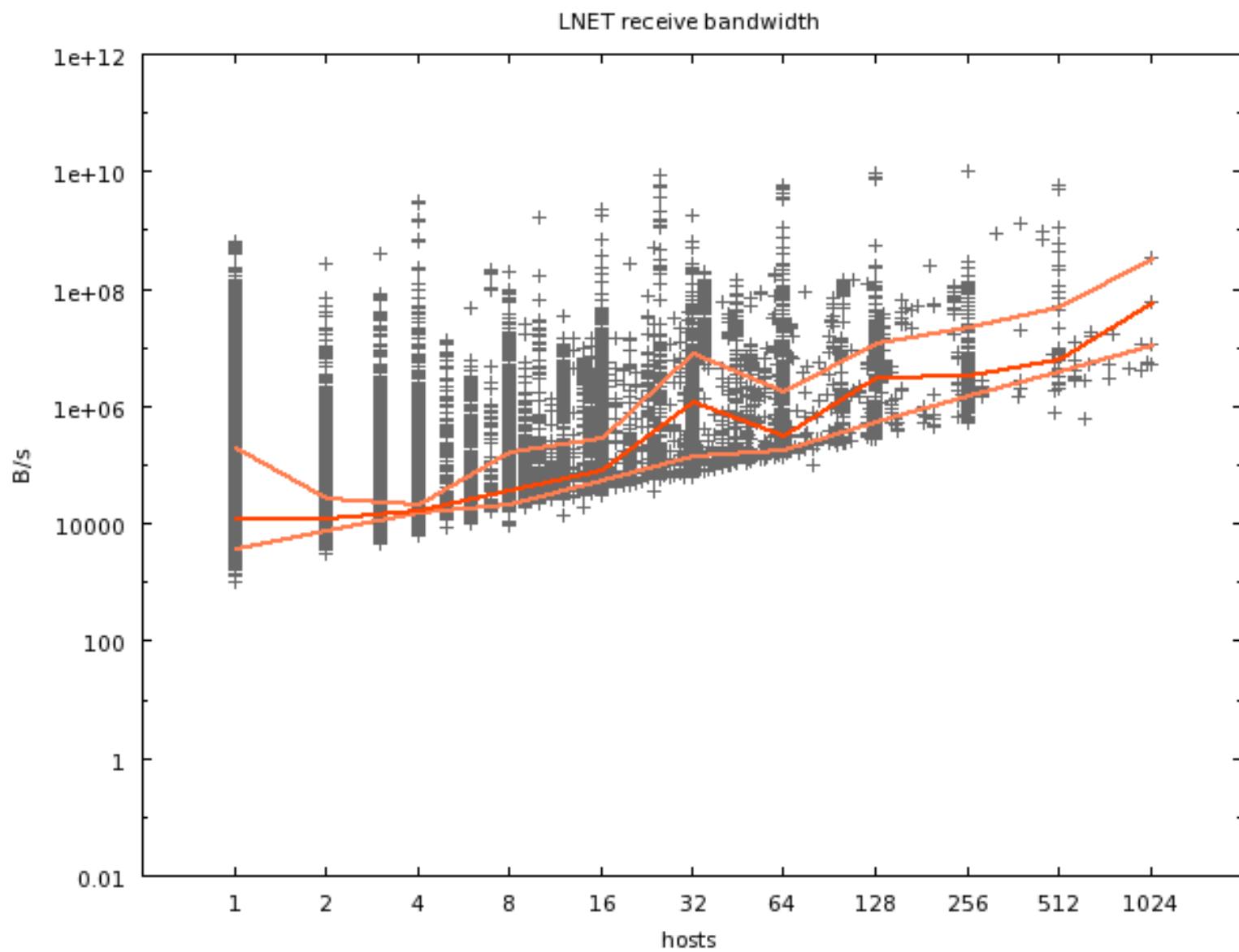
1 Ranger month: 60 GB

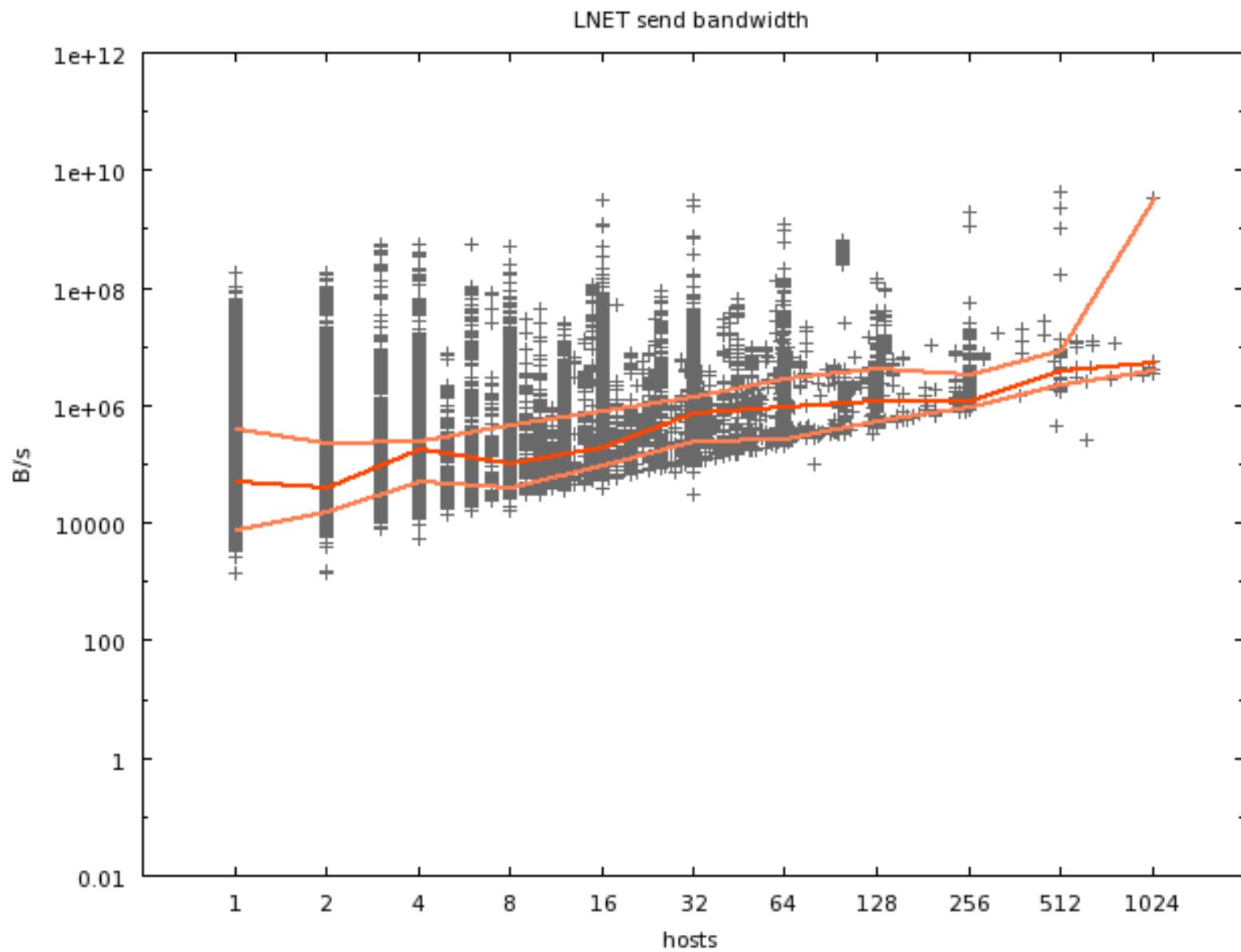


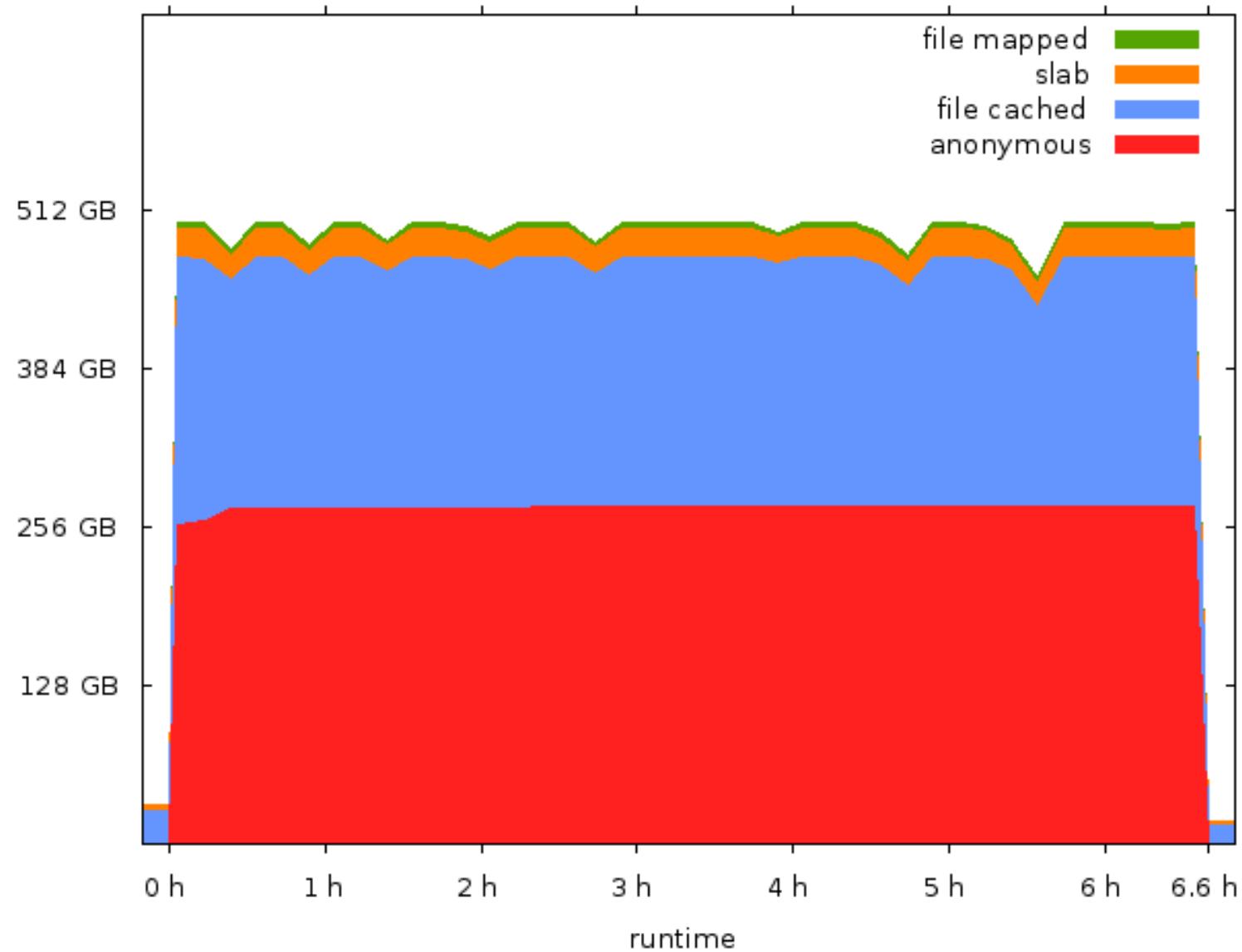
/scratch posix read bandwidth











Thanks!

https://github.com/jhammond/tacc_stats